USB ARMORY

PAST, PRESENT AND FUTURE

Andrea Barisani – Head of Hardware Security

F-Secure

$ whoami

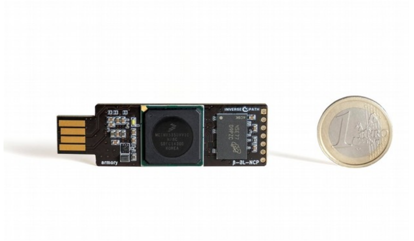
Andrea “lcars” Barisani

I am a

Founder of INVERSE PATH now part of

Breaking things since I got my first

Securing and much more since 2005.

Maker of the USB armory 

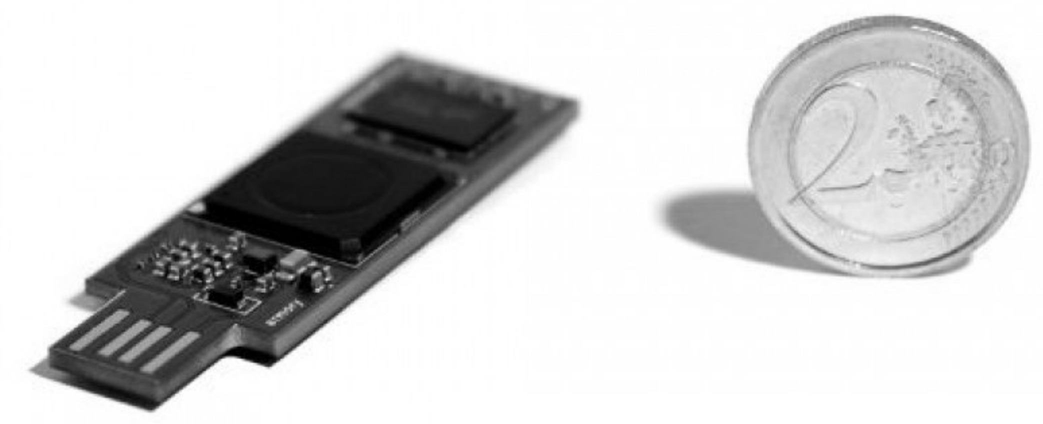
Speaker and trainer at BlackHat, CanSecWest, DEFCON, Hack In The Box, PacSec conferences among many others.

https://andrea.bio | @andreabarisani

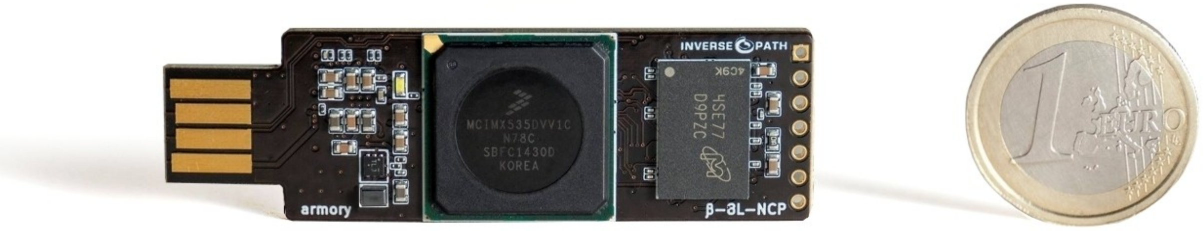
October 15th - Hack In The Box 2014 – Kuala Lumpur

Inverse Path Announce Armory SoC Project

posted on OCTOBER 3, 2014 by L33TDAWG



USB armory Mk I – Design goals



Compact USB powered device

Fast CPU and generous RAM

Secure boot

Standard connectivity over USB

Familiar developing/execution environment

Open design

USB armory Mk I - Specifications

NXP i.MX53 ARM® Cortex™-A8 800Mhz, 512MB RAM

USB host powered (<500 mA) device with compact form factor (65 x 19 x 6 mm)

ARM® TrustZone®, secure boot + storage + RAM

microSD card slot

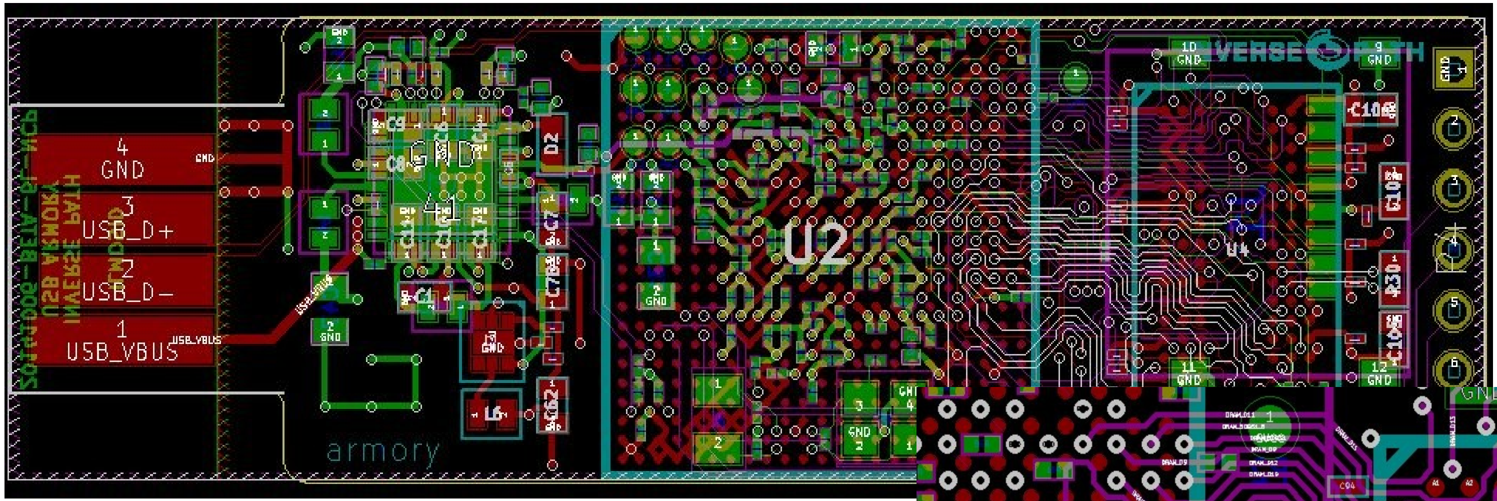
5-pin breakout header with GPIOs and UART, customizable LED (TZ)

Debian, Ubuntu, Arch Linux ARM, Genode OS (w/ TZ)

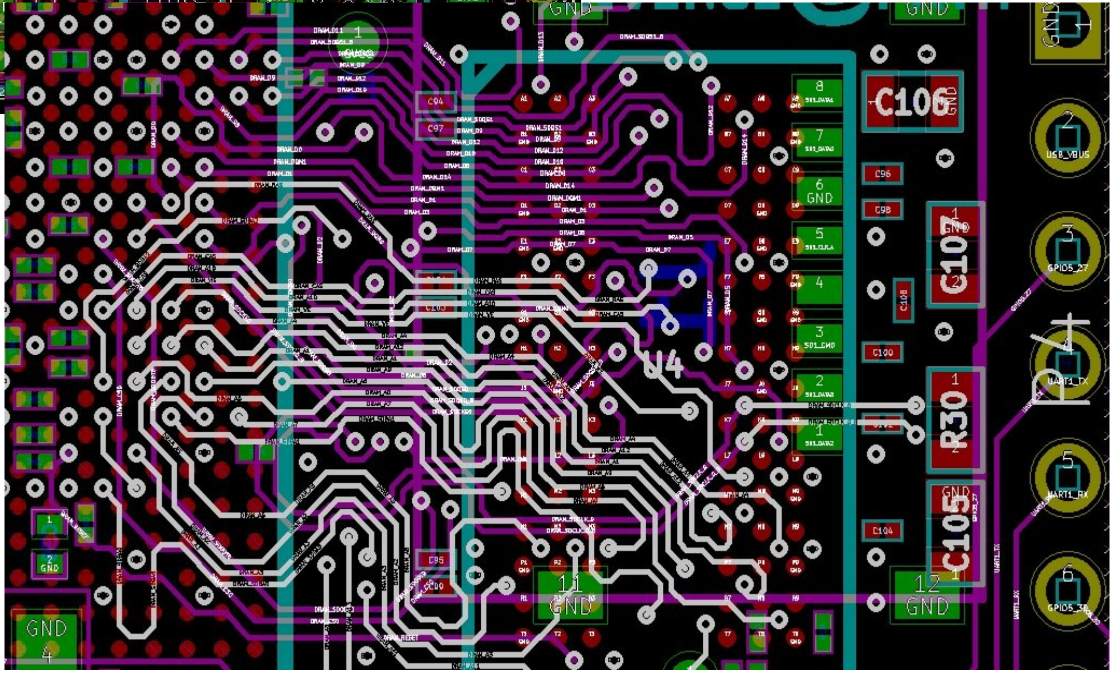
USB device emulation (CDC Ethernet, mass storage, HID, etc.)

Open Hardware & Software

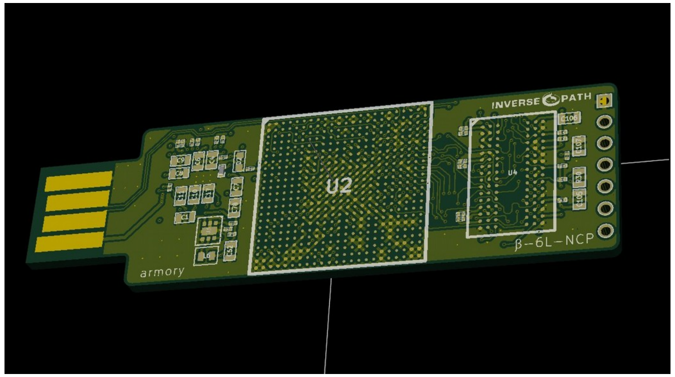
F-Secure



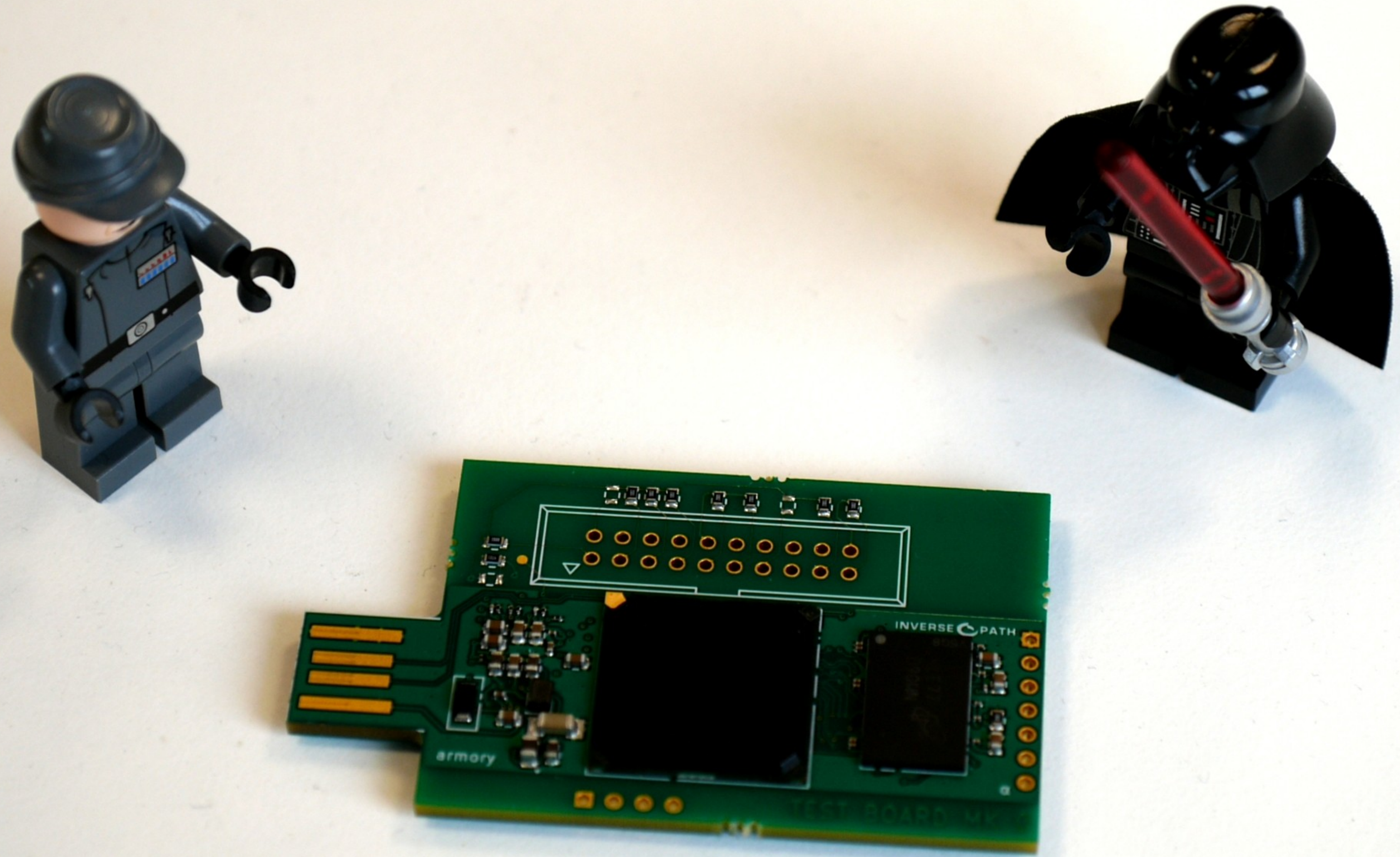
4 GND
3 USB_D+
2 USB_D−
1 USB_VBUS
U2
U4
armory
C106
C107
R30
C105

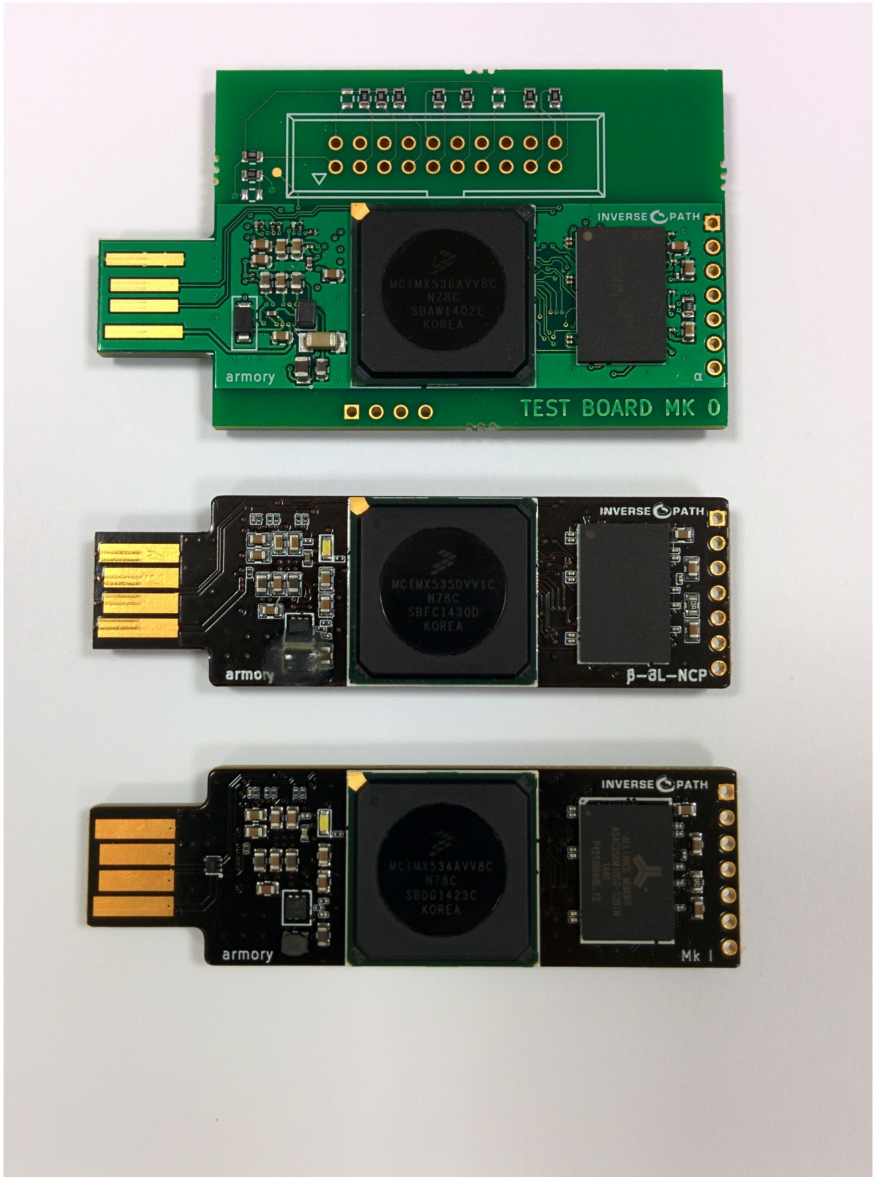


C106
C107
R30
C105
U4
GND
USB_VBUS
GPIO5_27
UART1_TX
UART1_RX
GPIO5_28



INVERSE PATH
U2
U4
armory
β-6L-NCP


INVERSE PATH
armory



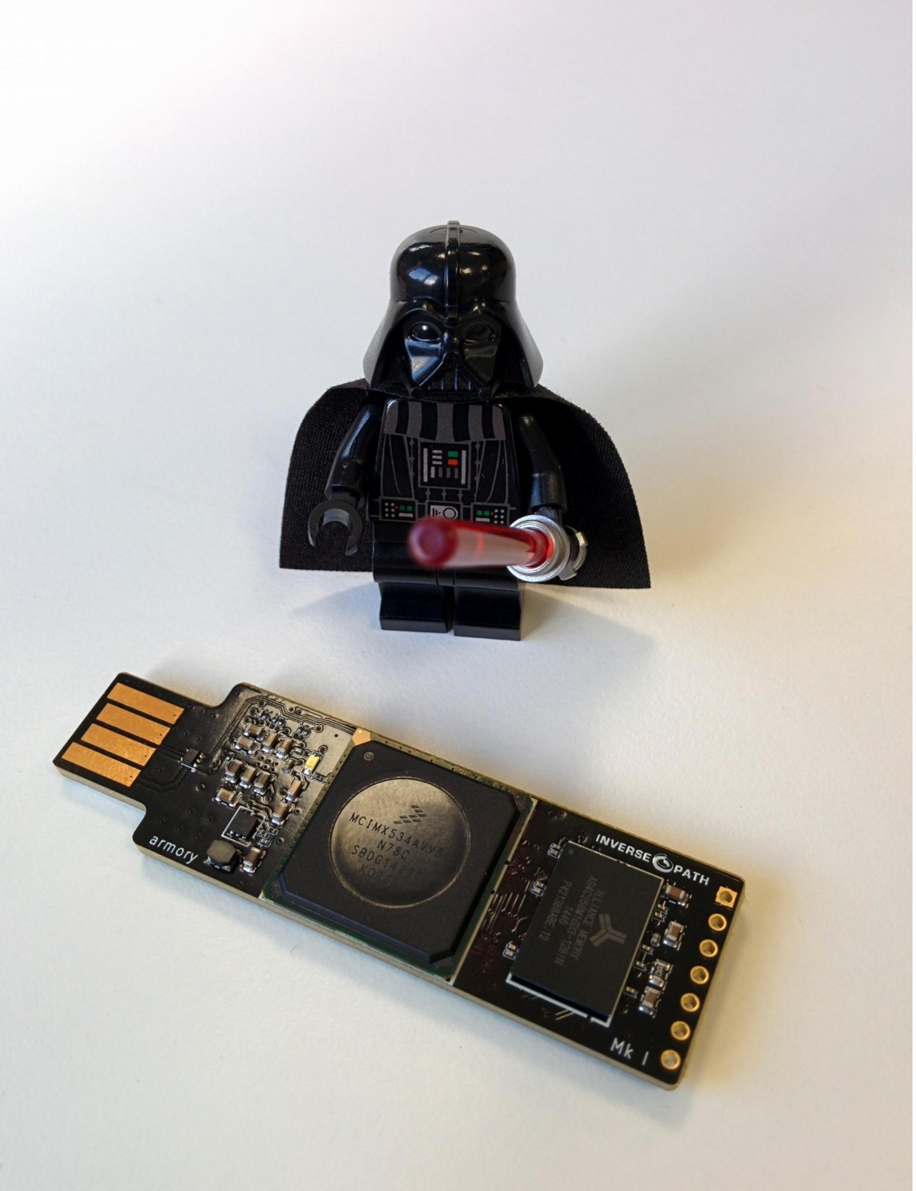
α

βs

8L-NOUSBH,8L, 8L-DDR-LDO, 8L-DDR-NCP

6L, 6L-DDR-LDO, 6L-DDR-NCP

Mk I



100% Made in Italy
(PCB manufacturing + assembly + enclosure)

F-Secure

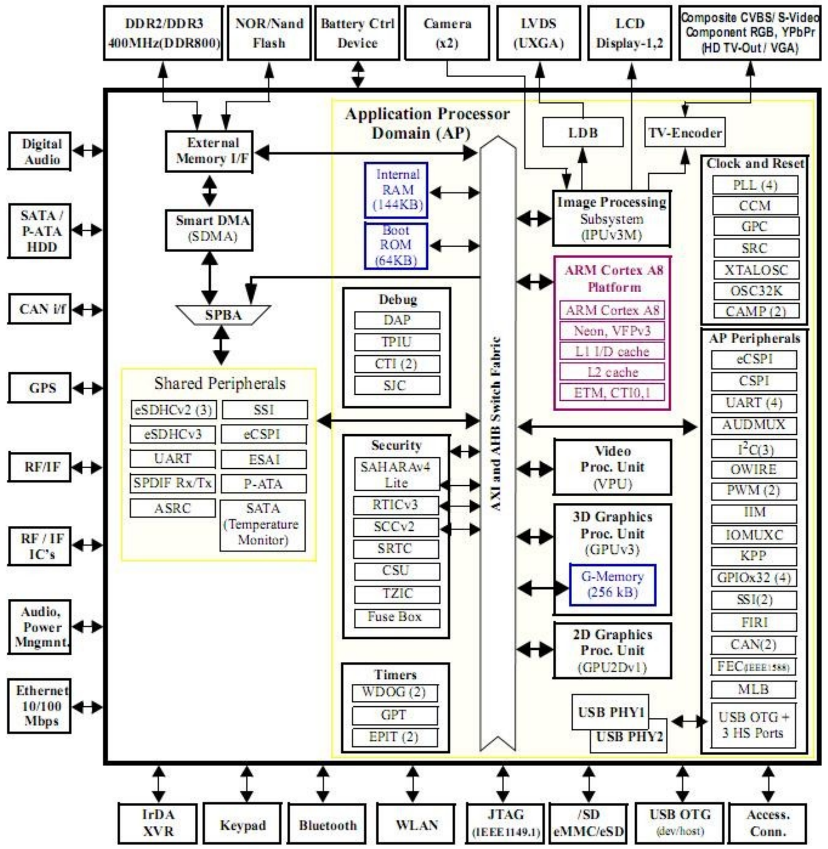
The USB armory runs Linux and is a personal server designed for security applications:

Password manager
Encrypted storage
Authentication token
Cryptocurrency wallet
Secure messaging
Hardware Security Module

Customers range from individual security researchers, security companies to large enterprises and government entities.

NXP - i.MX53

ARM® Cortex™-A8 800 Mhz



Hardware security features

High Assurance Boot (HAB 4.0.4)

Security Controller (SCCv2)

Cryptographic accelerator (SAHARAv4 LITE)

Run-time integrity checker (RTIC)

ARM® TrustZone®

NXP - HAB (secure boot)

High Assurance Boot (HABv4) enables boot image (bootloader) verification.

Up to four public keys (SRK) are used to generate a SHA256 hash for verification, the hash is fused on the SoC with a permanent, irreversible operation.

The main bootloader (e.g. U-Boot) image is signed using one of such four keys, such information, and additional metadata, is placed in a Command Sequence File (CSF) for parsing and authentication.

Initially supported only by NXP proprietary tools, we re-implemented their functionality as open source tools.

HAB tools

Hash generation

```
$ usbarmory_srktool -h
Usage: usbarmory_srktool [OPTIONS]
  -1 | --key1  <public key path>     SRK public key 1 in PEM format
  -2 | --key2  <public key path>     SRK public key 2 in PEM format
  -3 | --key3  <public key path>     SRK public key 3 in PEM format
  -4 | --key4  <public key path>     SRK public key 4 in PEM format
  -o | --hash  <output filename>     Write SRK table hash to file
  -O | --table <output filename>     Write SRK table to file
```

Bootloader signing

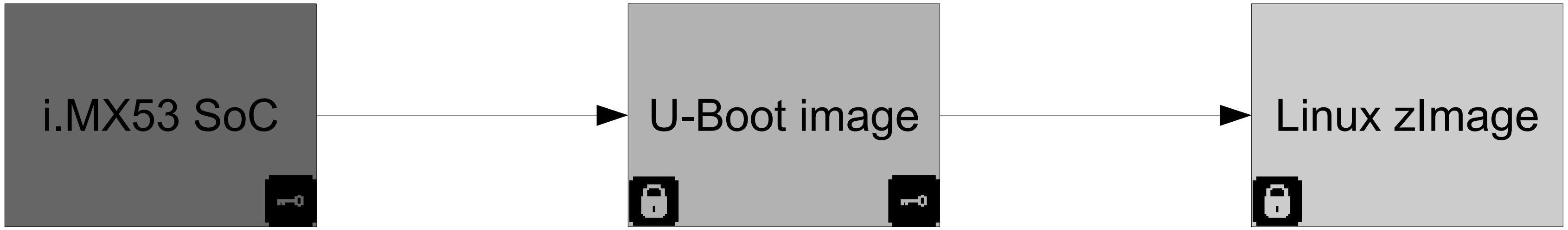
```
$ usbarmory_csftool -h
Usage: usbarmory_csftool [OPTIONS]
  -A | --csf_key <private key path>  CSF private key in PEM format
  -a | --csf_crt <public  key path>  CSF public  key in PEM format
  -B | --img_key <private key path>  IMG private key in PEM format
  -b | --img_crt <public  key path>  IMG public  key in PEM format
  -I | --table   <SRK table path>    Input SRK table (see usbarmory_srktool -O)
  -x | --index   <SRK key index>     Index for SRK key (1-4)
  -i | --image   <filename>          Image file w/ IVT header (e.g. u-boot.imx)
  -o | --output  <filename>          Write CSF to file
```

Verified Boot

The U-Boot bootloader supports cryptographic verification of signed kernel images. A public key can be embedded in the bootloader image to verify the Linux kernel.

```
make ARCH=arm EXT_DTB=pubkey.dtb
```

The chain of trust, up to the kernel image, authenticates all executed code with user controlled certificates.



NXP – Security Controller (SCCv2)

F-Secure®

The SCCv2 is a built-in hardware module that implements secure RAM and a dedicated AES cryptographic engine for encryption/decryption operations.

A device specific random 256-bit SCC key is fused in each SoC at manufacturing time, this key is unreadable and can only be used with the SCCv2 for AES encryption/decryption of user data.

The SCCv2 internal key is available only when Secure Boot (HAB) is enabled, otherwise the AES-256 NIST standard test key is used.

Useful to derive device-specific secrets for FDE.

We implemented, based on old Freescale code which we ported and upgrade, a userspace interface through a custom driver to allow its use.

NXP – Security Controller (SCCv2)

https://github.com/inversepath/mxc-scc2

```
$ sudo modprobe scc2
$ sudo modprobe scc2_aes

SCC2_AES: Secure State detected
```

Ruby example:

```
scc = File.open("/dev/scc2_aes", "r+")

# encryption
scc.ioctl(SET_MODE, ENCRYPT_CBC)
scc.ioctl(SET_IV, iv)

scc.write(plaintext)
ciphertext = scc.read(plaintext.size)

# decryption
scc.ioctl(SET_MODE, DECRYPT_CBC)
scc.ioctl(SET_IV, iv)

scc.write(ciphertext)
plaintext = scc.read(ciphertext.size)
```

INTERLOCK

https://github.com/inversepath/interlock

Open source file encryption front-end developed, but not limited to, usage with the USB armory.

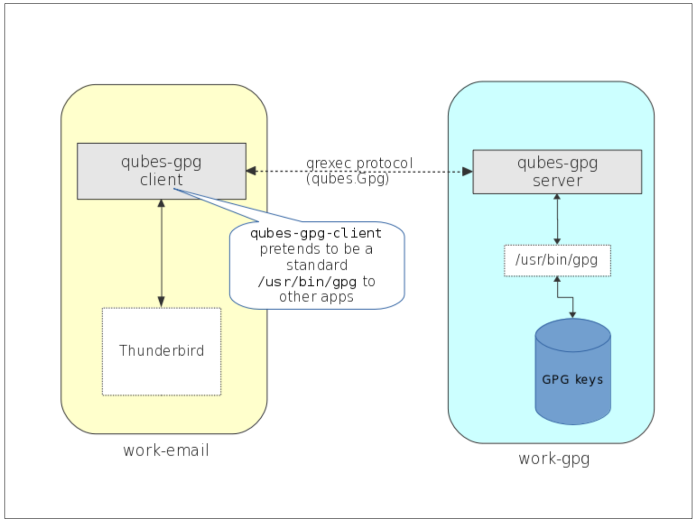
Provides a web accessible file manager to unlock/lock LUKS encrypted partition and perform additional symmetric/asymmetric encryption on stored files.

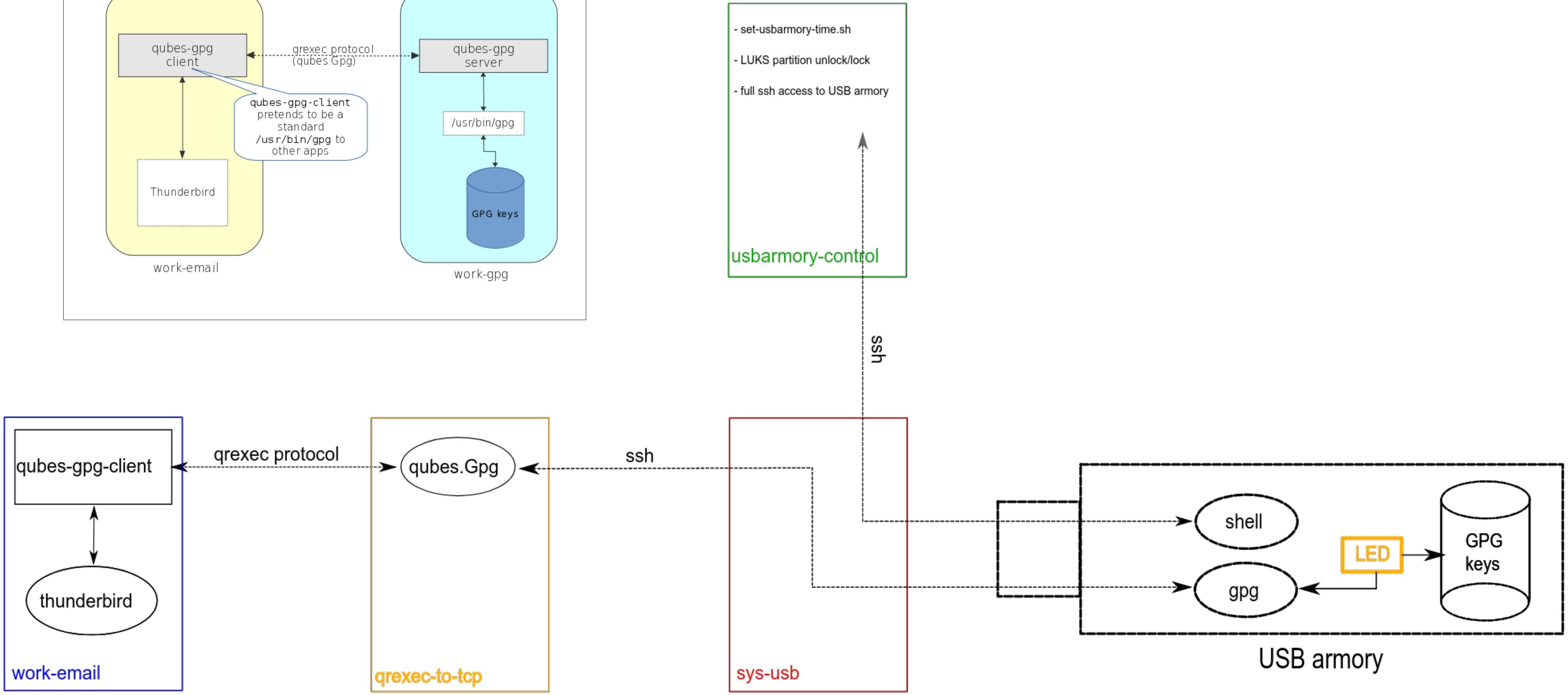
Takes advantage of disposable passwords.

Supported technologies: LUKS, OpenPGP, TOTP, Signal, SCCv2 for device specific keys.

DEMO

Qubes Split GPG server


qubes-gpg client
qrexec protocol (qubes.Gpg)
qubes-gpg server
qubes-gpg-client pretends to be a standard /usr/bin/gpg to other apps
/usr/bin/gpg
Thunderbird
GPG keys
work-email
work-gpg

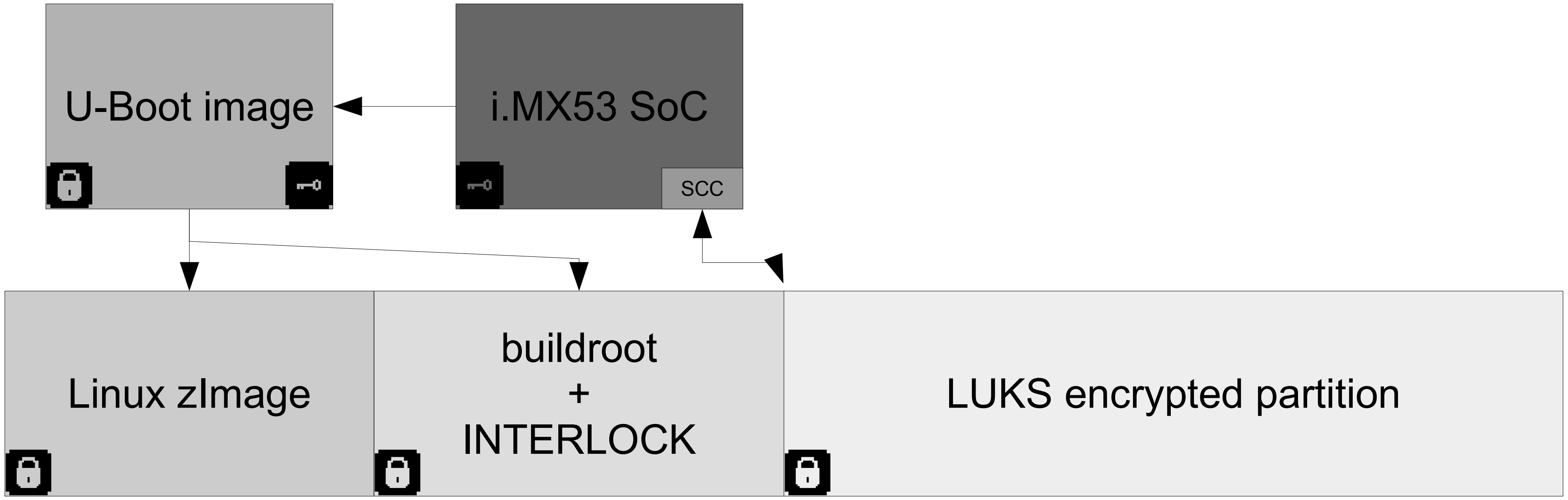


BUILDROOT

https://github.com/inversepath/usbarmory/tree/master/software/buildroot

Custom buildroot profiles allow compilation of bootloader, kernel, runtime environment and target application with an automatic cross-compilation process.

```
make BR2_EXTERNAL=${USBARMORY_GIT}/software/buildroot interlock_mark_one_defconfig
make BR2_EXTERNAL=${USBARMORY_GIT}/software/buildroot # yes, it's that easy!
```



OFFENSIVE USES

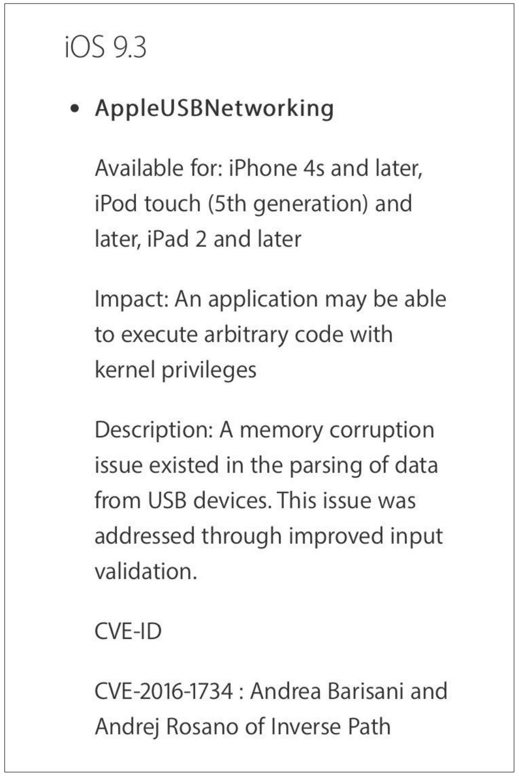
USB descriptors + drivers manipulation/fuzzing, passive sniffing, DNS hijacking and traffic diversion.

Some nice papers involving the USB armory:

Jeroen van Kessel, Nick Petros Triantafyllidis
"USB Armory as an Offensive Attack Platform"

Roland Schilling, Frieder Steinmetz
"USB devices phoning home"

Matthias Neugschwandtner, Anton Beitler, Anil Kurmus
"A Transparent Defense Against USB Eavesdropping Attacks"


iOS 9.3

- **AppleUSBNetworking**

  Available for: iPhone 4s and later, iPod touch (5th generation) and later, iPad 2 and later

  Impact: An application may be able to execute arbitrary code with kernel privileges

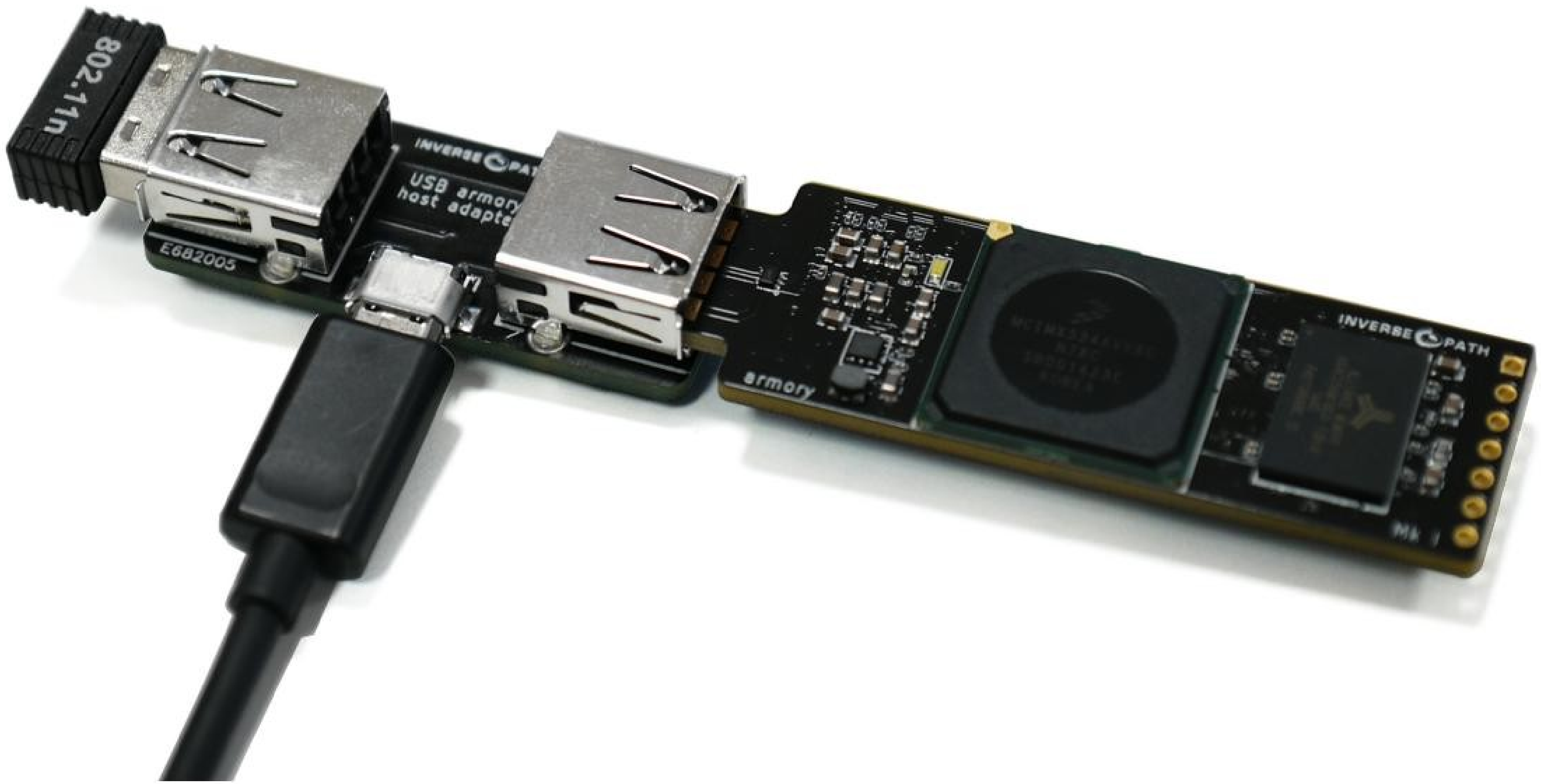
  Description: A memory corruption issue existed in the parsing of data from USB devices. This issue was addressed through improved input validation.

  CVE-ID

  CVE-2016-1734 : Andrea Barisani and Andrej Rosano of Inverse Path

HOST ADAPTER

F-Secure



INVERSE PATH

inversepath.com/usbarmory

HABv4 bypass

In 2017 Quarkslab discovered critical security vulnerabilities that affect HABv4 on the entire NXP i.MX series.

The issue was reported for the i.MX6, Inverse Path immediately investigated applicability to the i.MX53.

A X.509 parsing error (ERR010873 | CVE-2017-7932) and an SDP protection bypass (ERR01872 | CVE-2017-7936) allow arbitrary code execution on SoC in Closed configuration.

The findings prevent the secure operation of unattended setups while attended setups remain protected in case of device loss (but not tampering).

NXP did not release any P/N updates for the i.MX53.

https://github.com/inversepath/usbarmory/blob/master/software/secure_boot/Security_Advisory-Ref_QBVR2017-0001.txt

HABv4 bypass

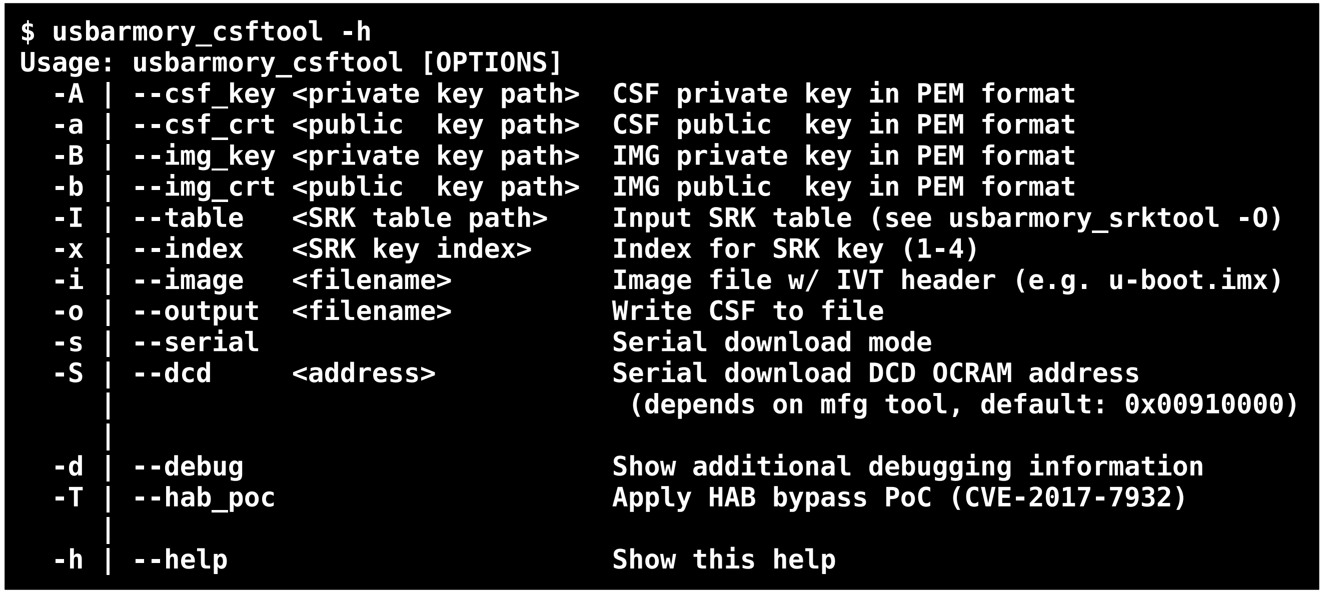
```
Timeline
========

2017-05-18: Quarkslab presents findings at the 2017 Qualcomm Mobile Security
            Summit [9], materials are not disclosed to the public at this time.
2017-05-30: Quarkslab communicates embargo period until 2017-07-18.
2017-05-30: Inverse Path proposes preliminary advisory release on 2017-06-05.
2017-06-05: Inverse Path releases preliminary advisory.
2017-06-06: added assigned CVE numbers.
2017-07-19: Quarkslab public release of findings [4].
2017-07-19: Inverse Path release of full advisory and i.MX53 PoC [6].
2017-07-27: added link to i.MX Community post that lists affected P/Ns.
```

The team at Inverse Path prioritized announcing the existence of the issue before the full advisory release, additionally developed and released a full PoC.

The `usbarmory_csftool` is the only Open Source implementation for HABv4 signing as well as the first and only exploitation tool ;-)

"Break your own product, and break it hard"
https://labsblog.f-secure.com/2017/07/19/break-your-own-product-and-break-it-hard/

HABv4 bypass



```
$ usbarmory_csftool -h
Usage: usbarmory_csftool [OPTIONS]
  -A | --csf_key <private key path>  CSF private key in PEM format
  -a | --csf_crt <public  key path>  CSF public  key in PEM format
  -B | --img_key <private key path>  IMG private key in PEM format
  -b | --img_crt <public  key path>  IMG public  key in PEM format
  -I | --table   <SRK table path>    Input SRK table (see usbarmory_srktool -O)
  -x | --index   <SRK key index>     Index for SRK key (1-4)
  -i | --image   <filename>          Image file w/ IVT header (e.g. u-boot.imx)
  -o | --output  <filename>          Write CSF to file
  -s | --serial                      Serial download mode
  -S | --dcd     <address>           Serial download DCD OCRAM address
     |                                (depends on mfg tool, default: 0x00910000)
     |
  -d | --debug                       Show additional debugging information
  -T | --hab_poc                     Apply HAB bypass PoC (CVE-2017-7932)
     |
  -h | --help                        Show this help
```

Publishing PoC code encourages further investigation and testing of issues among vendors or other affected parties; it promotes security research; and it empowers other skilled parties to further verify the scope and impact of vulnerabilities.

The most important and compelling reason to take this approach, however, is this: In scenarios where detailed technical information has already been made public, the lack of a working PoC does not, and should not, constitute any form of "protection."

USB armory Mk I

F-Secure

One of the smallest SBC in the world, met with outstanding demand from security researchers, businesses, OEMs, integrators and security companies.

The good

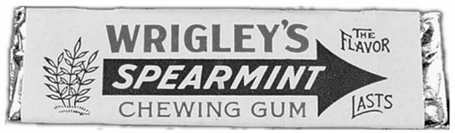
It wasn’t easy to fill the support gap left by NXP, but we did it until we hit the actual hardware and this resulted in several OSS contributions.

Form factor, priority on security and transparency, the incredible projects and use cases we never dreamed of.

Great research platform for all things (e.g. TrustZone).

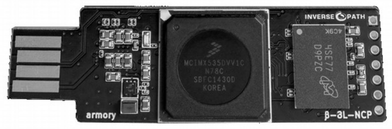
MACGYVER ARMORY

20th Century



DNA storage, conductor, sealing material, adhesive, stress relieve, nomnom
76mm x 19mm

21th Century



Open source hardware and software, ARM Cortex-A8 800MHz, 512MB RAM, microSD, USB 2.0 OTG, Ethernet/storage/UART/HID/etc device emulation,
65mm x 19mm

USB armory Mk I - coolest projects

USB armory Mk I - current status

The bad

- The microSD hinge is “challenging”.
- The PCB plug, in retrospect, was a bad call.
- Designing enclosure as an afterthought is a nightmare.
- We learned the hard way that NXP long term support does not entail security.
- Lack of built-in storage restricts provisioning scalability.
- Not ideal for general consumer applications.

The future

We are actively working on the **USB armory Mk II** to continue our support for this class of product and improve it. This is what we are trying to achieve:

The microSD hinge replacement with a push/pull slot.

Real USB plugs, plug + socket for integrated host adapter.

Enclosure design right from the beginning.

Full internal and third party security audit for HABv4 and chain of trust.

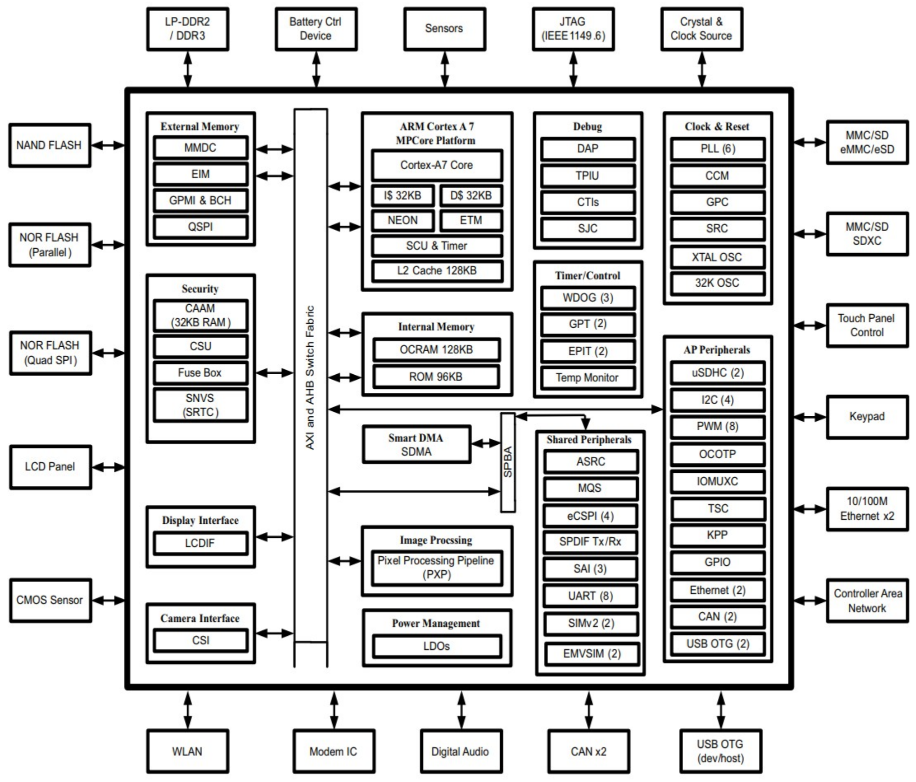
Addition of built-in eMMC storage and external crypto authenticator.

Bluetooth communication.

https://github.com/inversepath/usbarmory/wiki/Mk-II-Roadmap

NXP - i.MX6UL

ARM® Cortex™-A7 528 Mhz (900 Mhz with i.MX6ULL option but w/o CAAM and BEE)



Hardware security features

High Assurance Boot (HAB 4.2.6)

Cryptographic accelerator and assurance module (CAAM)

Bus Encryption Engine (BEE - OTF AES)

Secure Non-Volatile Storage (SNVS)

Run-time integrity checker (RTIC)

ARM® TrustZone®

NXP - Secure Non-Volatile Storage (SNVS)

The SNVS feature relies on the OTPMK which cannot be read directly as it can only be used via the SoC internal Cryptographic Accelerator and Assurance Module (CAAM), when secure booted.

The SNVS feature can be summarized as follows:

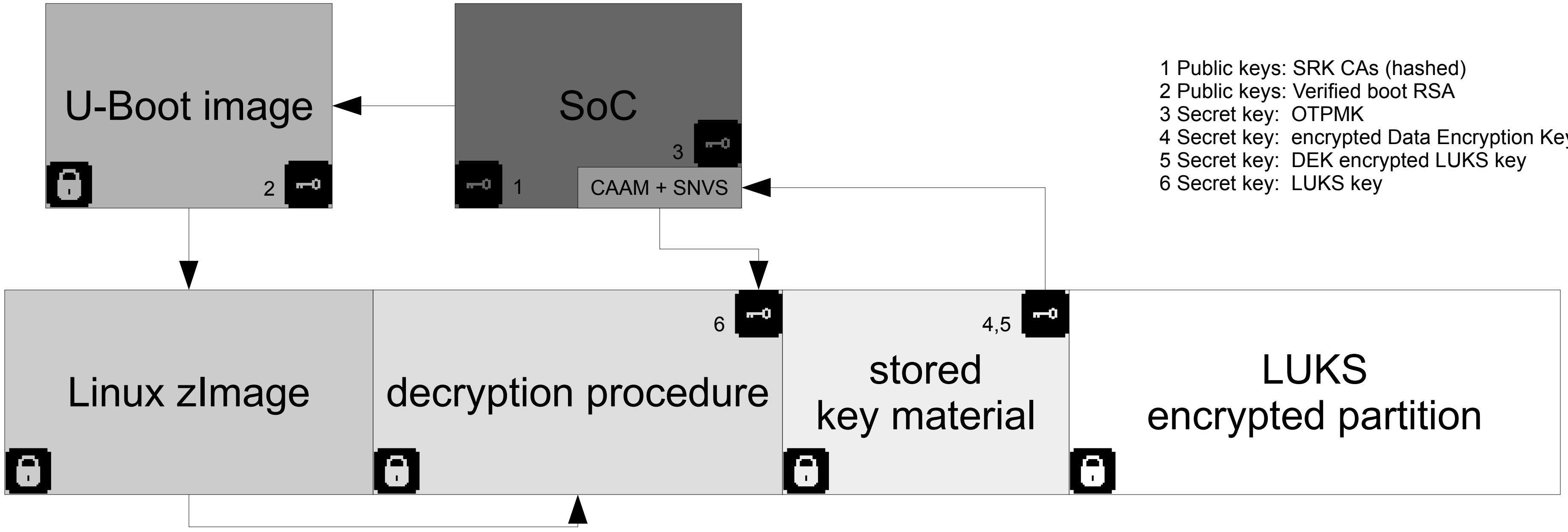
A random 256-bit blob encryption key (DEK) is generated.

The blob encryption key is used to encrypt the desired data via the CAAM AES-CCM function, providing confidentiality and integrity protection.

The blob encryption key is AES-ECB encrypted with a key derived from the OTPMK, using a Single-step Key-Derivation Function, resulting in the DEK blob.

The HAB secure boot sequence, or runtime environment, can directly support authenticated decryption of arbitrary data blobs (including the bootloader image).

Full chain of trust example - i.MX6



1 Public keys: SRK CAs (hashed)
2 Public keys: Verified boot RSA
3 Secret key: OTPMK
4 Secret key: encrypted Data Encryption Key (DEK)
5 Secret key: DEK encrypted LUKS key
6 Secret key: LUKS key

🔑 key material

🔒 authenticated + encrypted

▶ SoC	authenticates U-Boot
▶ U-Boot	authenticates Linux
▶ Linux	uses SVNS decrypted key material to unlock the encrypted partition

NXP CAAM + SNVS driver

https://github.com/inversepath/caam-keyblob

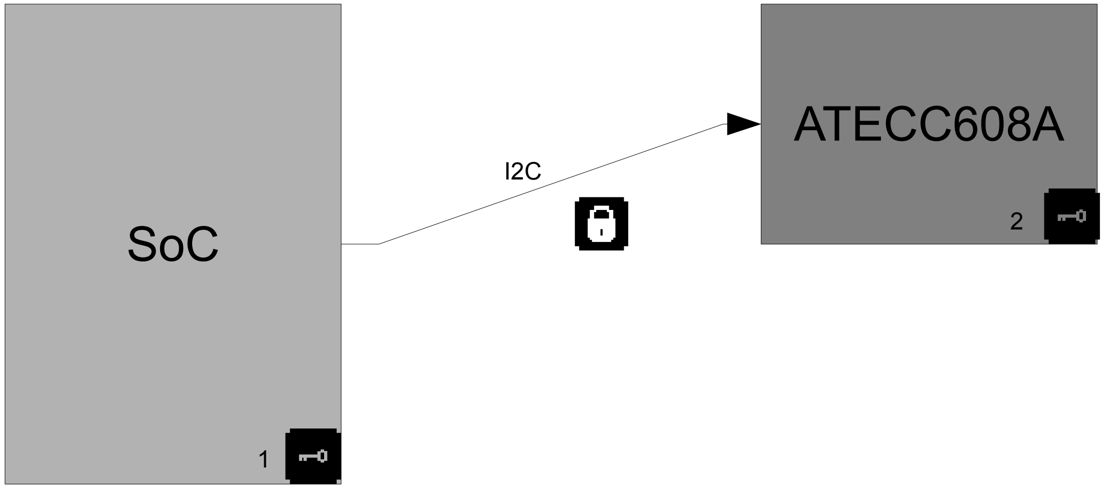
```
$ sudo modprobe caam_keyblob

caam_keyblob: Secure State detected
```

Go userspace implementation:

```
$ caam_tool enc dek.bin dek_blob.bin
caam_tool: encrypting 32 bytes from dek.bin
caam_tool: caam_kb_data &{Text:0x49c000 TextLen:32 Blob:0x4a0000 BlobLen:80 Keymod:0x48c010 KeymodLen:16}
caam_tool: encrypted 80 bytes to dek.bin

$ caam_tool dec dek.bin dek_blob.bin
caam_tool: decrypting 80 bytes from dek_blob.bin
caam_tool: caam_kb_data &{Text:0x478000 TextLen:32 Blob:0x474000 BlobLen:80 Keymod:0x412140 KeymodLen:16}
caam_tool: decrypted 32 bytes to dek.bin
```

Now supported by INTERLOCK for LUKS key, TLS certificate protection and AES cipher support (`”hsm”: “caam-keyblob:luks,tls,cipher”`).

Rollback protection + external keyring



1 Secret keys: ReadKey, WriteKey

2 Slots (16x) for key, certificates or data
High Endurance Monotonic Counters (2x)
OTPs (512-bit)

🔑 key material

🔒 authenticated + encrypted

The SoC can establish a secure session with the ATECC608A, using safely stored read and write keys, certificates or data.

This allows secure key/certificate access or use, additionally two High Endurance Monotonic Counters can be used for rollback protection.

Provides an additional hardware keyring for (partial) mitigation of further HAB issues.

i.MX6UL - Security audit

Against Silicon Revision 1.2 and HAB 4.1 or greater, meaning P/Ns “AB” or greater, with patched HABv4.

Completed as an internal + third party security audit for HABv4 as well as our buildroot chain of trust implementation.

Conclusions

No further issues have been identified in the patched boot ROM.

Freescale kernel module issues (invalid error values, NULL pointer exceptions, various operational errors), all resolved in our own caam-keyblob driver implementation.

U-Boot issues...

Security advisory - IPVR2018-0001

Multiple techniques allow execution of arbitrary code, bypassing secure boot and/or verified boot.

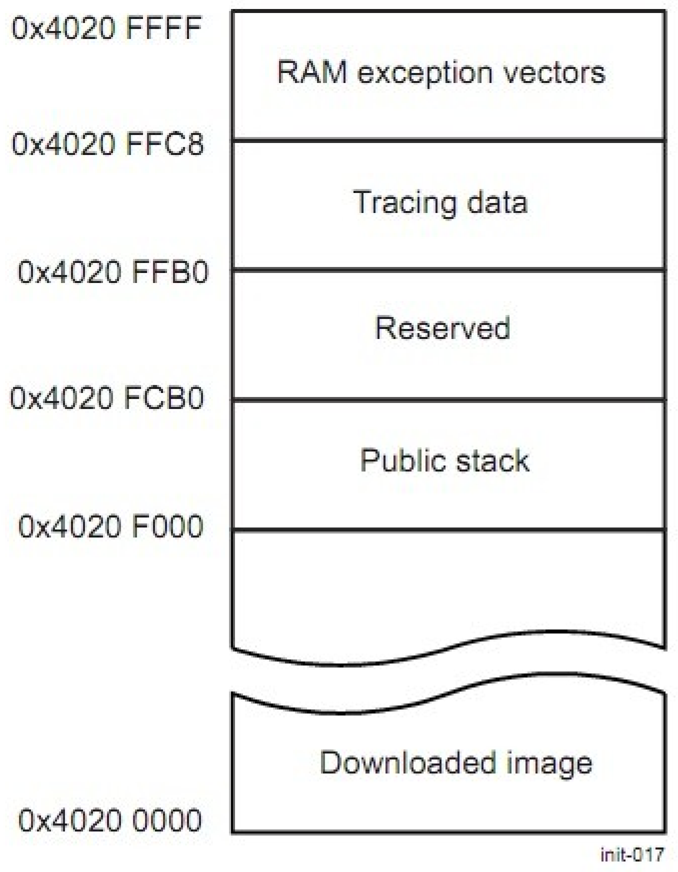
U-Boot lacks any automatic memory allocation protection in relation to its own code location in memory.

CVE-2018-18440

Lack of boundary checks in filesystem image load.

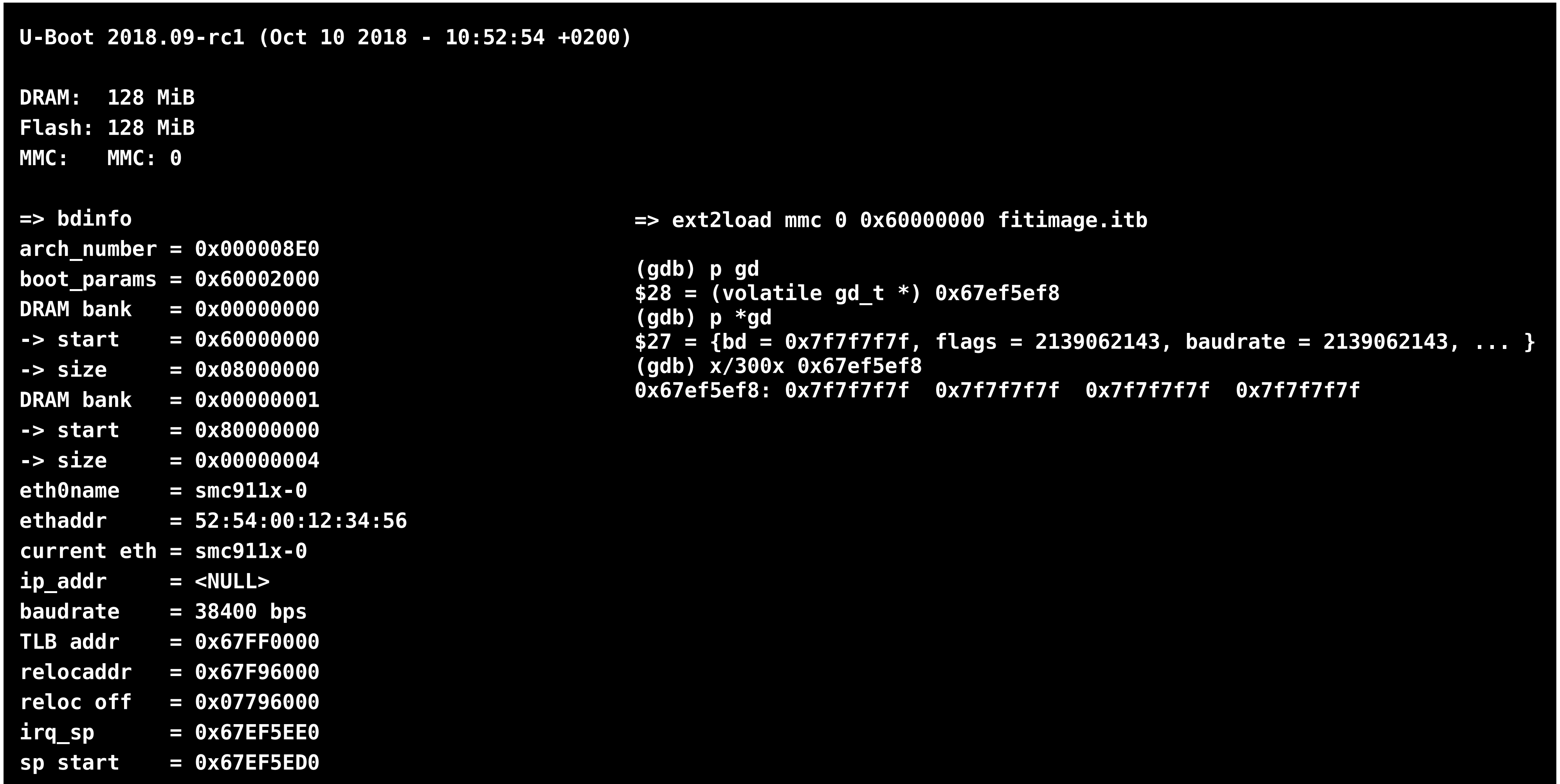
CVE-2018-18439

Lack of boundary checks in network image boot.



https://github.com/inversepath/usbarmory/blob/master/software/secure_boot/Security_Advisory-Ref_IPVR2018-0001.txt

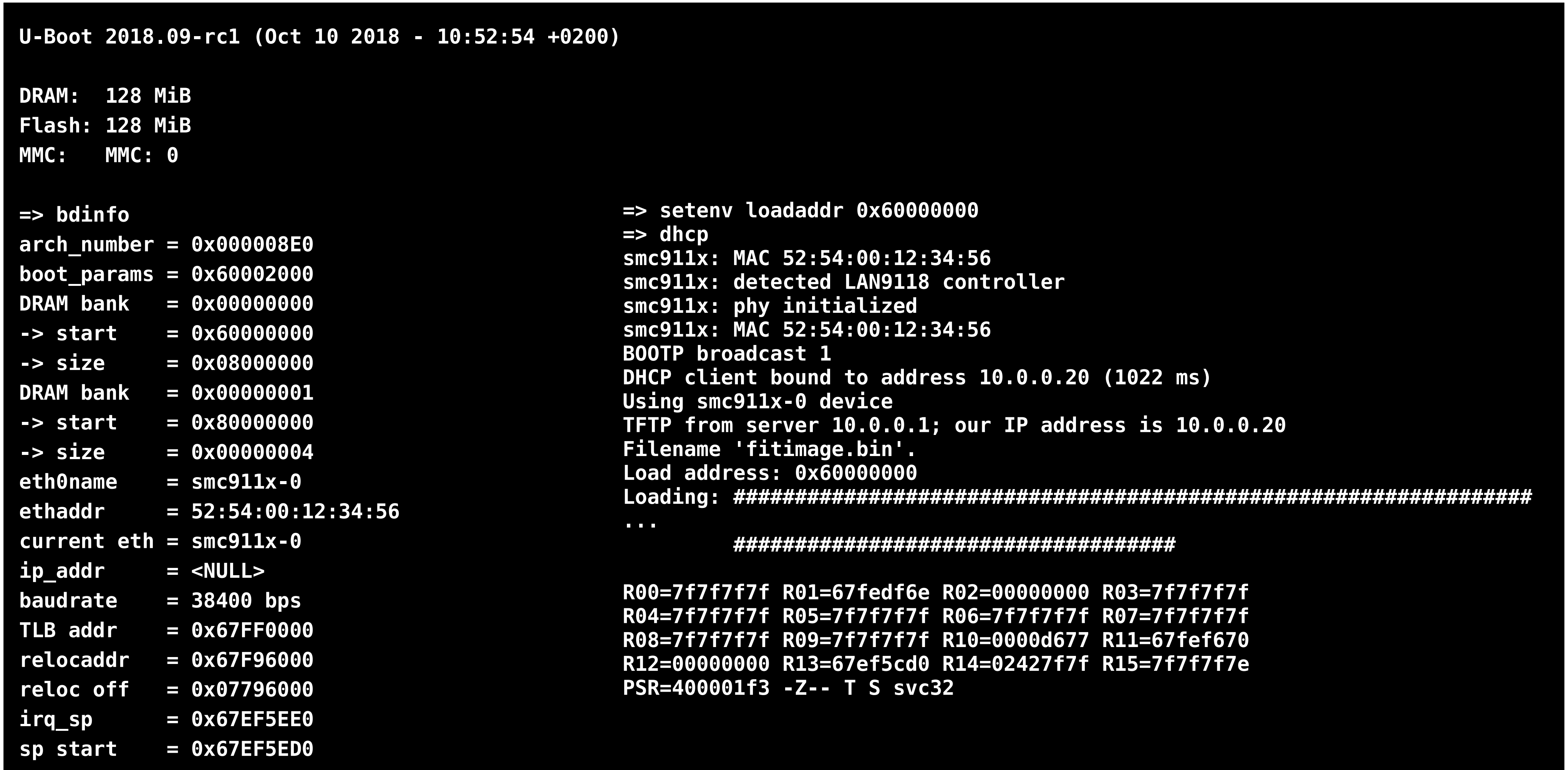
CVE-2018-18440



```
U-Boot 2018.09-rc1 (Oct 10 2018 - 10:52:54 +0200)

DRAM:  128 MiB
Flash: 128 MiB
MMC:   MMC: 0

=> bdinfo
arch_number = 0x000008E0
boot_params = 0x60002000
DRAM bank   = 0x00000000
-> start    = 0x60000000
-> size     = 0x08000000
DRAM bank   = 0x00000001
-> start    = 0x80000000
-> size     = 0x00000004
eth0name    = smc911x-0
ethaddr     = 52:54:00:12:34:56
current eth = smc911x-0
ip_addr     = <NULL>
baudrate    = 38400 bps
TLB addr    = 0x67FF0000
relocaddr   = 0x67F96000
reloc off   = 0x07796000
irq_sp      = 0x67EF5EE0
sp start    = 0x67EF5ED0
```

```
=> ext2load mmc 0 0x60000000 fitimage.itb

(gdb) p gd
$28 = (volatile gd_t *) 0x67ef5ef8
(gdb) p *gd
$27 = {bd = 0x7f7f7f7f, flags = 2139062143, baudrate = 2139062143, ... }
(gdb) x/300x 0x67ef5ef8
0x67ef5ef8: 0x7f7f7f7f  0x7f7f7f7f  0x7f7f7f7f  0x7f7f7f7f
```

CVE-2018-18439



```
U-Boot 2018.09-rc1 (Oct 10 2018 - 10:52:54 +0200)

DRAM:  128 MiB
Flash: 128 MiB
MMC:   MMC: 0

=> bdinfo
arch_number = 0x000008E0
boot_params = 0x60002000
DRAM bank   = 0x00000000
-> start    = 0x60000000
-> size     = 0x08000000
DRAM bank   = 0x00000001
-> start    = 0x80000000
-> size     = 0x00000004
eth0name    = smc911x-0
ethaddr     = 52:54:00:12:34:56
current eth = smc911x-0
ip_addr     = <NULL>
baudrate    = 38400 bps
TLB addr    = 0x67FF0000
relocaddr   = 0x67F96000
reloc off   = 0x07796000
irq_sp      = 0x67EF5EE0
sp start    = 0x67EF5ED0
```

```
=> setenv loadaddr 0x60000000
=> dhcp
smc911x: MAC 52:54:00:12:34:56
smc911x: detected LAN9118 controller
smc911x: phy initialized
smc911x: MAC 52:54:00:12:34:56
BOOTP broadcast 1
DHCP client bound to address 10.0.0.20 (1022 ms)
Using smc911x-0 device
TFTP from server 10.0.0.1; our IP address is 10.0.0.20
Filename 'fitimage.bin'.
Load address: 0x60000000
Loading: #################################################################
...
         ####################################

R00=7f7f7f7f R01=67fedf6e R02=00000000 R03=7f7f7f7f
R04=7f7f7f7f R05=7f7f7f7f R06=7f7f7f7f R07=7f7f7f7f
R08=7f7f7f7f R09=7f7f7f7f R10=0000d677 R11=67fef670
R12=00000000 R13=67ef5cd0 R14=02427f7f R15=7f7f7f7e
PSR=400001f3 -Z-- T S svc32
```

Workarounds

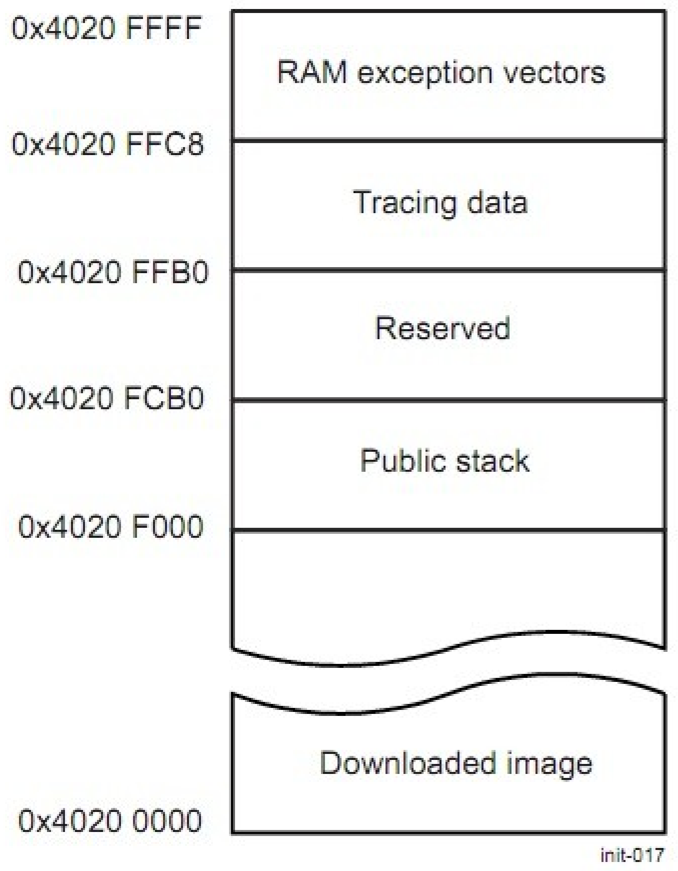
It must be emphasized these two cases only represent two possible occurrences of such architectural limitation, other U-Boot image loading functions are extremely likely to suffer from the same validation issues.

CVE-2018-18440

The optional bytes argument can be passed to all load commands to restrict the maximum size of retrieved data. A value consistent with memory regions mapping can be passed.

CVE-2018-18439

Disable all network loading commands.



https://github.com/inversepath/usbarmory/blob/master/software/secure_boot/Security_Advisory-Ref_IPVR2018-0001.txt

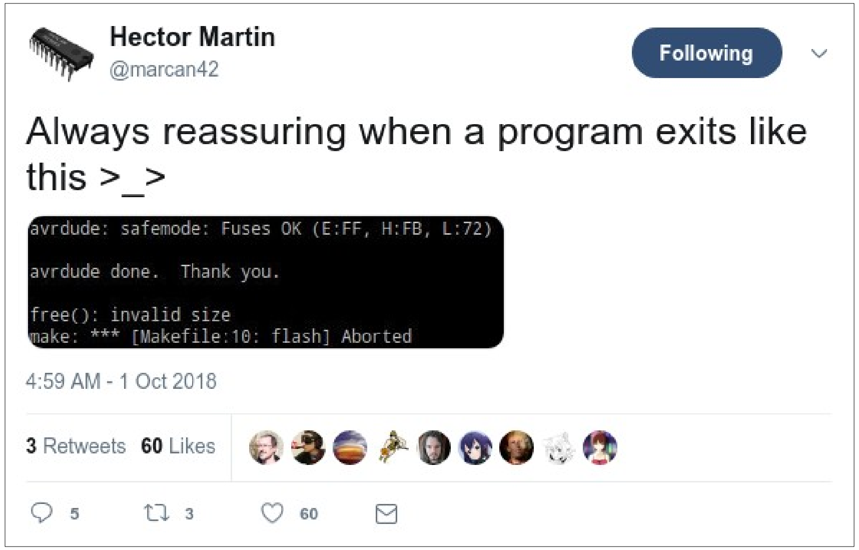
One-Time-Programmable (OTP) fuses

The available tools (e.g. U-Boot fuse command) or frameworks (Linux NVMEM) provide raw, low level, access.

The NXP documentation is often inconsistent, erroneous, imprecise in describing the SoC fusemap.

Some Linux NVMEM drivers are buggy and do Not account for addressing gaps, leading to Inconsistent read/write operations.

We need better tools!



Hector Martin
@marcan42
Following

Always reassuring when a program exits like this >_>

```
avrdude: safemode: Fuses OK (E:FF, H:FB, L:72)

avrdude done.  Thank you.

free(): invalid size
make: *** [Makefile:10: flash] Aborted
```

4:59 AM - 1 Oct 2018

3 Retweets 60 Likes

5 3 60

```
# Example of OTP fusing within U-Boot.
#
# syntax: fuse prog [-y] <bank> <word> <hexval> [<hexval>...] - program 1 or
#         several fuse words, starting at 'word' (PERMANENT)
=> fuse prog -y 1 0x1 0xaa
=> fuse prog -y 3 0x1 0xbb 0xcc 0xdd 0xee 0xff 0xaa 0xbb 0xcc 0xdd 0xee 0xff
=> fuse prog -y 3 0xc 0xaa 0xbb 0xcc 0xdd 0xee 0xff 0xaa 0xbb 0xcc 0xdd 0xee
=> fuse prog -y 3 0x17 0xff 0xaa 0xbb 0xcc 0xdd 0xee 0xff 0xaa 0xbb
```

Introducing the crucible.

https://github.com/inversepath/crucible

CRUCIBLE

A tool that provides user space support for reading, and writing, One-Time-Programmable (OTP) fuses of System-on-Chip (SoC) application processors.

```
Usage: crucible [options] [read|blow] [fuse/register name] [value]
  -Y    do not prompt for confirmation (DANGEROUS)
  -b int
        value base/format (2,10,16)
  -f string
        YAML fuse maps directory (default "fusemaps")
  -l    list available fuse maps
  -m string
        processor model (default "IMX6UL")
  -n string
        NVMEM device (default "/sys/bus/nvmem/devices/imx-ocotp0/nvmem")
  -r string
        reference manual revision
  -s    use syslog, print ony result value to stdout
```

CRUCIBLE - Fusemaps

The crucible relies on fusemaps in YAML format, which map registers and fuses names with their driver-specific addressing, accounting for eventual gaps.

```
processor: <string>          # processor model
reference: <string>          # reference manual number (for P/N revision match)
driver: <string>             # Linux driver name
                             #
gaps:                        # gap definitions
  <string>:                  #   name of first register after gap
    read: <bool>             #     applies to read operation
    write: <bool>            #     applies to write operation
    len: <uint32>            #     gap length in bytes
                             #
registers:                   # register definitions
  <string>:                  #   register name
    bank: <uint32>           #     bank index
    word: <uint32>           #     word index
    fuses:                   #     individual OTP fuse definitions
      <string>:              #       fuse name
        offset: <uint32>     #         fuse offset within register word
        len: <uint32>        #         fuse length in bits
```

Currently supported drivers

Vendor	Mod	Linux driver	Read	Write	Fusemap
NXP	i.MX53	nvmem-imx-iim	yes	no	Yes
NXP	i.MX6Q	nvmem-imx-ocotp	yes^	yes	No
NXP	i.MX6SL	nvmem-imx-ocotp	yes^	yes	No
NXP	i.MX6SX	nvmem-imx-ocotp	yes^	yes	No
NXP	i.MX6UL	nvmem-imx-ocotp	yes^	yes	Yes
NXP	i.MX7D	nvmem-imx-ocotp	yes^	yes	No

CRUCIBLE - Fusemaps

```
---
# i.MX6 UltraLite Applications Processor Reference Manual
# iMX6ULRM Rev. 1, 04/2016
#
processor: IMX6UL
reference: 1

# On the IMX6UL a gap is present between OTP Bank5 Word7 (0x21B_C6F0) and OTP
# Bank6 Word0 (21B_C800).
#
# The nvmem-imx-ocotp driver does not handle addressing gaps between OTP banks,
# the fusemap supports gap information specifically to work this problem around
# and ensure correct reads (writes are unaffected). Such driver limitation
# however does not allow for the entire fusemap to be read as its maximum size
# is computed without accounting for the gaps.
#
# For this specific fusemap banks 14 and 15, while defined, are not available
# for read operation because of the aforementioned driver issues.
#
# The gap definition below is required to ensure correct read operations for
# all registers beyond the gap.
#
driver: nvmem-imx-ocotp
gaps:
  OCOTP_ROM_PATCH0:
    read: true
    len: 0x100
```

CRUCIBLE - i.MX6UL example

Blow a fuse

```
$ sudo crucible -m IMX6UL -r 1 -b 16 blow MAC1_ADDR 0x001f7b1007e3
IMX6UL ref:1 op:blow addr:0x88 off:0 len:48 val:0xe307107b1f000000
```

Read a fuse

```
$ sudo crucible -m IMX6UL -r 1 -b 16 read MAC1_ADDR
IMX6UL ref:1 op:read addr:0x88 off:0 len:48 val:0x001f7b1007e3
```

Read a fuse (minimal output for batch operations)

```
$ sudo crucible -s -m IMX6UL -r 1 -b 16 read MAC1_ADDR
001f7b1007e3
```

```
registers:
  OCOTP_LOCK:
    bank: 0
    word: 0
    fuses:
      TESTER_LOCK:
        offset: 0
        len: 2
      BOOT_CFG_LOCK:
        offset: 2
        Len: 2
...
  OCOTP_MAC0:
    bank: 4
    word: 2
    fuses:
      MAC1_ADDR:
        offset: 0
        len: 48
  OCOTP_MAC1:
    bank: 4
    word: 3
  OCOTP_MAC:
    bank: 4
    word: 4
```

Summary of new releases

USB armory Mk II - Roadmap and development progress
https://github.com/inversepath/usbarmory/wiki/Mk-II-Roadmap

caam-keyblob - CAAM + SNVS driver
https://github.com/inversepath/caam-keyblob

INTERLOCK with CAAM support
https://github.com/inversepath/interlock

crucible - OTP fusing tool
https://github.com/inversepath/crucible

i.MX6UL - Buildroot profile for embedded INTERLOCK distribution
https://github.com/inversepath/usbarmory/tree/master/software/buildroot

U-Boot security advisory - CVE-2018-18439, CVE-2018-18440
https://github.com/inversepath/usbarmory/blob/master/software/secure_boot/Security_Advisory-Ref_IPVR2018-0001.txt

Thank you!

Questions?

Andrea Barisani

Founder
Inverse Path | inversepath.com
andrea@inversepath.com

Head of Hardware Security
F-Secure | f-secure.com
andrea.barisani@f-secure.com